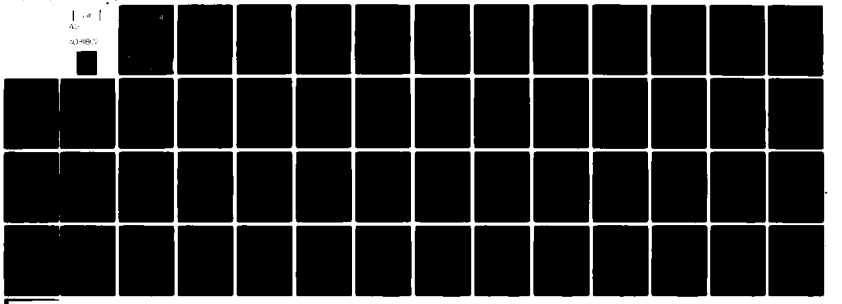
AD-A081 802 COLORADO UNIV AT BOULDER DEPT OF COMPUTER SCIENCE F/G 12/1
DETERMINING FEASIBILITY OF A SET OF NONLINEAR INEQUALITY CONSTR--ETC(U)
FEB 80 R B SCHNABEL DAAG29-79-C-0023
UNCLASSIFIED CU-CS-172-80 ARO-15074.11-M NL



END
DATE FILMED
4-80
DTIC

DETERMINING FEASIBILITY OF A SET OF NONLINEAR INEQUALITY CONSTRAINTS

by

Robert B. Schnabel
Department of Computer Science
University of Colorado at Boulder
Boulder, Colorado

CU-CS-172-80 February, 1980

INTERIM TECHNICAL REPORT
U. S. ARMY RESEARCH OFFICE
CONTRACT NO. DAAG29-79-C-0023

Presented at the Tenth International Symposium on Mathematical Programming, Montreal, August 1979 and the Fall 1979 ORSA-TIMS meeting, Milwaukee, October 1979.

THE FINDINGS IN THIS REPORT ARE NOT TO BE CONSTRUED AS AN OFFICIAL DEPARTMENT OF THE ARMY POSITION, UNLESS SO DESIGNATED BY OTHER AUTHORIZED DOCUMENTS.

We acknowledge U. S. Army Research support under grant DAAG29-78-G-0046 and National Science Foundation support under grant no. MCS77-02194.

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER: CU-CS-172-80	2. GOVT ACCESSION NO.
3. RECIPIENT'S CATALOG NUMBER	
4. TITLE (and Subtitle): Determining Feasibility of a Set of Nonlinear Inequality Constraints	5. TYPE OF REPORT & PERIOD COVERED
	6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s): Robert B. Schnabel	8. CONTRACT OR GRANT NUMBER(s): DAAG29-79-C-0023, DAAG29-78-G-0046, MCS77-02194
9. PERFORMING ORGANIZATION NAME AND ADDRESS: Department of Computer Science, University of Colorado at Boulder, Boulder, Colorado 80309	10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS: U. S. Army Research Office, Post Office Box 12211, Research Triangle Park, NC 27709	12. REPORT DATE: Feb 1980
	13. NUMBER OF PAGES: 47
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office)	15. SECURITY CLASS. (of this report): unclassified
	15a. DECLASSIFICATION/DOWNGRADING SCHEDULE: NA

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

NA

18. SUPPLEMENTARY NOTES

The findings in this report are not to be construed as an official Department of the Army position, unless so designated by other authorized documents.

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

feasible point, inequality constraints, linearly constrained optimization

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

We present a new class of algorithms for determining whether there exists a point $x \in R^n$ satisfying the nonlinear inequality constraints $c_i(x) \leq 0$, $i = 1, \ldots, m$, subject perhaps to satisfying linear inequality constraints $\ell_j(x) \leq 0$, $j = 1, \ldots, k$ which are known to be feasible. Our algorithm consists of solving a sequence of linearly constrained optimization problems, using a sequence of objective functions $\phi(x,p)$ which are at least twice continuously differentiable, and which are generated by monotonically increasing the value of the non-negative parameter p. It is shown that in almost all cases, once p

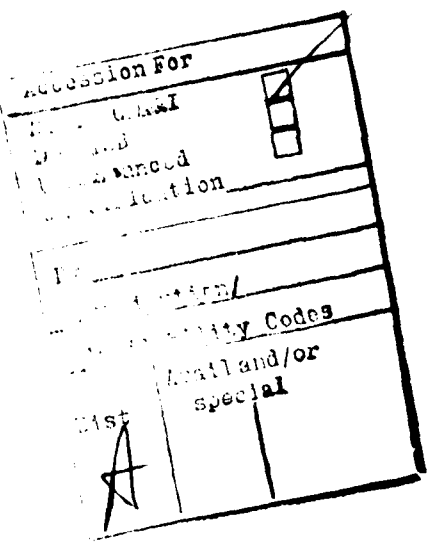
DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

Unclassified

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

reaches or exceeds some finite value, that the solution to the linearly constrained optimization problem either is a feasible point, or establishes the infeasibility of the set of constraints. Computational results are presented in which the algorithm performs satisfactorily on feasible and on infeasible systems.

TABLE OF CONTENTS :



TABLES

FIGURES

Abstract

We present a new class of algorithms for determining whether there exists a point $x \in R^n$ satisfying the nonlinear inequality constraints $c_i(x) \leq 0$, $i = 1, \ldots, m$, subject perhaps to satisfying linear inequality constraints $\ell_j(x) \leq 0$, $j = 1, \ldots, k$ which are known to be feasible. Our algorithm consists of solving a sequence of linearly constrained optimization problems, using a sequence of objective functions $\phi(x,p)$ which are at least twice continuously differentiable, and which are generated by monotonically increasing the value of the non-negative parameter p. It is shown that in almost all cases, once p reaches or exceeds some finite value, that the solution to the linearly constrained optimization problem either is a feasible point, or establishes the infeasibility of the set of constraints. Computational results are presented in which the algorithm performs satisfactorily on feasible and on infeasible systems.

1. Introduction

In this paper, we present and analyze a new class of algorithms for determining whether a system of linear and nonlinear inequality constraints has a feasible point. The general problem we consider is

$$\begin{aligned} &\text{Given } D \subseteq R^n,\ c_i:\ D \to R,\ i = 1, \ldots, m \\ &\text{Does there exist any } x \in \text{ such that} \\ &\quad c_i(x) \le 0,\ i = 1, \ldots, m? \end{aligned} \tag{1.1}$$

The m functions $c_i(x)$ are called constraint functions, or simply constraints. If the answer to (1.1) is yes, the set of constraints $\{c_i\}$ is said to be feasible, otherwise it is said to be infeasible. These definitions are made formally in Definition 1.1.

Problem (1.1) has many applications. Our original interest came from the static software validation project at the University of Colorado at Boulder, where (1.1) arises when determining satisfiability of path constraints (see e.g., [3]). In this case, many of the constraints are linear. Since problem (1.1) is easily solved in the linear case, it is advantageous to treat linear constraints separately from nonlinear ones. Also, in most applications of (1.1) it is necessary that the algorithm actually return a feasible x when the answer to (1.1) is yes. Thus the form of problem (1.1) we will address is:

$$\begin{aligned} &\text{Given } D \subseteq R^n, \\ &\qquad \text{nonlinear constraints } c_i : D \to R,\ i = 1, \ldots, m \\ &\qquad \text{linear constraints } \ell_j:\ D \to R,\ j = 1, \ldots, k \\ &\text{I) Determine whether there exist any } x \in D \text{ such that} \\ &\qquad \ell_j(x) \le 0,\ j = 1, \ldots, k \\ &\quad \text{(and if so:)} \end{aligned} \tag{1.2}$$

II) Either

a) Find an $x \in D$ such that $c_i(x) \leq 0$, $i = 1, \ldots, m$

subject to $\ell_j(x) \leq 0$, $i = 1, \ldots, k$

Or

b) Determine that no such x exists

Part I of problem (1.2) is simply phase I of linear programming, and there exist many reliable codes to solve it (see e.g., [4]). Thus our research is concerned solely with the solution of part II of (1.2) in the case when the linear constraints are feasible. It should be noted that our theoretical results remain valid and our computer algorithm performs just as well in the case k = 0, i.e., no linear constraints.

The reader may notice from the wording of problem (1.2) that we have attempted to construct a global algorithm i.e., one which solves (1.2) over the entire domain D, as opposed to a local algorithm, i.e, one which solves (1.2) in a neighborhood of the starting point. While we realize that a foolproof global algorithm for such a nonlinear problem is impossible we have developed our algorithm with a global solution in mind, and in practice have been rather successful. We have also assumed virtually no special structure, inclucing concavity or convexity, on the nonlinear constraints $c_i(x)$. In the applications we have seen, it is reasonable to assume that each c_i is twice continuously differentiable. Thus, the computer program described in this paper makes use of this assumption. However, the theoretical results of this paper do not even require c_i continuous. Other than this we make no assumption about the nonlinear constraints.

There does not appear to be much previous work on problem (1.2),

especially in the generality in which we are interested. A paper of Robinson [11] discusses a local method for finding a feasible point, assuming one starts close enough to one. Several other people, including Bertsekas [1] and Charalambous and Conn [2] have developed algorithms which can be applied to problem (1.2). The first approach would require optimization subject to nonlinear inequality constraints, the second, optimization of an objective function which is not continuously differentiable. Our approach differs strongly in that its optimization problems involve twice continuously differentiable objective functions, and only linear constraints.

The remainder of this paper is organized as follows. In Section 2 we motivate and present a new class of algorithms for solving problem (1.2) part ii. It involves a "weighting function" which is only required to satisfy some very general properties. In Section 3 we prove a number of theoretical properties of our class of algorithms, which show that they work successfully in almost all cases. Section 4 discusses the choice of the weighting function in practice. Section 5 describes other aspects of our computer algorithm, and presents some computational results. In Section 6 we summarize our work, and discuss the directions of our continuing research in this area.

We conclude this section with a formal definition of the concepts of feasibility and infeasibility, which were described in the opening paragraph. In addition, we introduce the terms strictly feasible, strictly infeasible and exactly feasible, which will be important in our discussions in Sections 3 and 4. Informally, a set of constraints is strictly feasible if it has a feasible point where in addition no constraint is equal to zero; it is strictly infeasible if it is

infeasible and does not come arbitrarily close to feasibility. It is exactly feasible if it is feasible but not strictly feasible.

Definition 1.1 Let $D \subseteq R^n$, and let $c_i : D \to R$, $i = 1, \ldots, m$. The set of constraints $\{c_i\}$ is said to be

feasible if there exists $x \in D$ such that $c_i(x) \leq 0$, $i = 1, \ldots, m$

infeasible if $\{c_i\}$ is not feasible, i.e., for all $x \in D$,

$$\max_{1 \leq i \leq m} \{c_i(x)\} > 0$$

strictly feasible if there exists $\varepsilon < 0$, $x \in D$ such that $c_i(x) \leq \varepsilon$, $i = 1, \ldots, m$

strictly infeasible if there exists $\delta > 0$ such that for all $x \in D$, $\max_{1 \leq i \leq m} \{c_i(x)\} \geq \delta$

exactly feasible if $\{c_i\}$ is feasible but not strictly feasible.

Given an $\varepsilon < 0$, $\{c_i\}$ is said to be

strictly ε-feasible if there exists $x \in D$ such that $c_i(x) \leq \varepsilon$, $i = 1, \ldots, m$

Given a $\delta > 0$, $\{c_i\}$ is said to be

within δ of feasibility if there exists $x \in D$ such that $c_i(x) \leq \delta$, $i = 1, \ldots, m$.

Note that if the domain D is closed and bounded, then a set of constraints which is infeasible must be strictly infeasible. On a computer, it is reasonable to assume that the domain is closed, with each component of x bounded in absolute value by the largest floating point number on the machine. Thus, it is reasonable to assume in practice that all infeasible problems are strictly infeasible.

2. The new class of algorithms: a weighting function-penalty approach

In this section, we introduce our new class of algorithms for solving problem (1.2), part II (henceforth called (1.2II)). Because it is somewhat complex, we motivate it by first describing an algorithm for a simpler problem. Our final algorithm is built around this simpler one.

Suppose first we wanted to solve problem (1.2II) with only one nonlinear constraint, i.e., m = 1. One can do this using well-known techniques, by simply solving the linearly constrained optimization problem

$$\begin{aligned} &\min_{x \in D} \ c_1(x) \\ &\text{subject to } \ell_j(x) \le 0, \ j = 1, \ldots, k \end{aligned} \tag{2.1}$$

If at the minimum x_* of (2.1), $c_1(x_*)$ is nonpositive, then the set of constraints $\{c_1, \ell_1, \ldots, \ell_k\}$ is feasible, otherwise it is infeasible. Notice, however, that if we desire a global answer to the feasibility problem (i.e., over all $x \in D$), then we will require the global minimum to (2.1). Also, if $c_1(x)$ is not convex, then our linearly constrained optimization routine cannot assume convexity.

A computer program to solve (2.1) under these conditions is, perhaps surprisingly, the key piece of software in our algorithm to solve the more general feasibility problem (1.2II). This is because our algorithm for solving (1.2II) turns out to consist of solving a series of problems of form (2.1). Furthermore, if we can successfully find the global minimum to the problems of form (2.1), then it turns out that we can find the global solution to (1.2II). (Otherwise we at least solve (1.2II) over whatever domain our linearly constrained

optimization algorithm is actually able to find its minima on.) Thus, the global aspect of our original problem is reduced to a somewhat more manageable framework.

Our algorithm for solving (2.1) is discussed in Section 5. In brief, we use the best existing techniques for finding a local minimum to (2.1) (see e.g., [6 , 7 , 8]), modified if necessary in the face of nonconvexity, and then restart in some other region of the domain if either curvature information accumulated during the local algorithm, or a sampling of points, indicate that a lower minimum may be found there. The routine has been quite successful in our tests.

Now consider the general case of problem (1.2II) with $m > 1$. It can also be reduced to the solution of a linearly constrained optimization problem in at least two ways. While they are not quite what we want, they lead the way to our algorithm. The first way is to solve

$$\begin{aligned} &\min_{x \in D} \quad \phi_1(x) = \sum_{i=1}^{m} c_i(x)^+ \\ &\text{subject to } \ell_j(x) \le 0,\ j = 1, \ldots, k \end{aligned} \tag{2.2}$$

where

$$c_i(x)^+ = \begin{cases} c_i(x) & c_i(x) > 0 \\ 0 & c_i(x) \le 0. \end{cases}$$

If the minimum value of ϕ_1 is zero, then (1.2II) is feasible, otherwise is is infeasible. The second way is to solve the "minimax" problem

$$\begin{aligned} &\min_{x \in D} \quad \phi_2(x) = \max_{1 \le i \le m} c_i(x) \\ &\text{subject to } \ell_j(x) \le 0,\ j = 1, \ldots, k. \end{aligned} \tag{2.3}$$

If the minimum value of ϕ_2 is nonpositive, then (1.2II) is feasible, otherwise it is infeasible.

The problem with both of these approaches is that they involve objective functions ϕ_1 and ϕ_2 which are not continuously differentiable. Standard software for linearly constrained optimization, including ours, does not work on such problems, and modifying it to be successful is difficult. However, (2.2) especially suggests a related idea which is the basis of our algorithm: to solve a problem of the form

$$\begin{aligned} &\min_{x \in D} \quad \phi(x) = \sum_{i=1}^{m} w(c_i(x)) \\ &\text{subject to } \ell_j(x) \leq 0, \; i = 1, \ldots, k \end{aligned} \tag{2.4}$$

where $w: R \to R$ is a <u>weighting function</u> which penalizes positive constraint values and rewards negative constraint values. The type of weighting function we will use is drawn in Figure 2.1, and formally defined in Definition 2.1. The most obvious example is $w(y) = e^y - 1$, but many other possibilities are also discussed in Section 4.

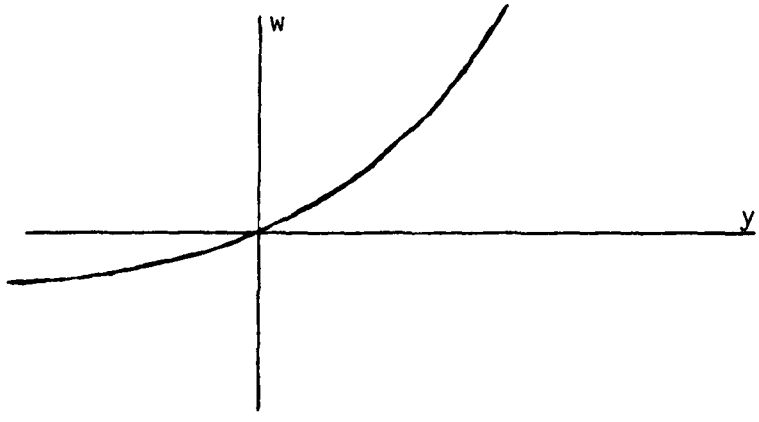


Figure 2.1 - A weighting function

<u>Definition 2.1</u>. A weighting function is a function $w : R \to R$ which obeys:

1) w is twice continuously differentiable.

2) $w(0) = 0$.

3) w is monotonically increasing (i.e., $w'(y) > 0$ for all $y \in R$).

4) w is strictly convex (i.e., $w''(y) > 0$ for all $y \in R$).

5) there exists finite $\ell < 0$ such that $\lim_{y \to -\infty} w(y) = \ell$.

6) $\lim_{y \to +\infty} w(y) = +\infty$

Note that property 6 is actually implied by the other properties.

Since w is twice continuously differentiable, problem (2.4) can be solved using our linearly constrained optimization software. Now let us see what information its solution gives us about the original feasibility problem (1.2II). Let us call the point at which (2.4) attains its minimum x_*. If $\phi(x_*) > 0$ then, just as in problems (2.2) and (2.3), (1.2II) must be infeasible. This is simply because if a feasible point $\bar{x}$ exists, then $\phi(\bar{x}) \leq 0$ by properties 2 and 3 of w, and so $\phi(x_*)$ would have to be negative. However, if $\phi(x_*) < 0$ then there are two possibilities: x_* may be feasible, or some constraints may be positive and others negative at x_*. In the final case, the solution to (2.4) does not give us the solution to feasibility problem (1.2II).

These possibilities are summarized below:

Let x_* be the solution to (2.4) where $w: R \to R$ obeys Definition 2.1. Then exactly one of the following is true:

1. $\phi(x_*) > 0$: this implies (1.2II) is infeasible.
2. $\phi(x_*) \leq 0$ and x_* is feasible: a feasible point to (1.2II) has been found.
3. $\phi(x_*) \leq 0$ and x_* is infeasible: (1.2II) may be feasible or infeasible.

Only the third case is unsatisfactory. To remedy it, we make one final addition to our process: we solve a <u>sequence</u> of problems

of form (2.4), using a sequence of weighting functions obeying Definition 2.1. These weighting functions successively penalize constraint violation more and more, and reward constraint satisfaction less and less. In this manner, they attempt to make any feasible point more attractive to problem (2.4) than any infeasible point. A representative sequence of such weighting functions is drawn in Figure 2.2; it is shown how to easily construct such sequences in Lemma 2.2. The key results of this paper (in Section 3) then show that by solving such a sequence of problems (2.4), we eventually end up with a solution x_* to (2.4) which falls into Case 1 or 2 above, instead of Case 3. Thus, we solve the feasibility problem (1.2II). (In practice, at least over the domain on which we solve the minimization problems.) Intuitively, this may also be viewed as saying that as our weighting function becomes shaped more like a backward "L," problem (2.4) comes to resemble problems (2.2) and (2.3).

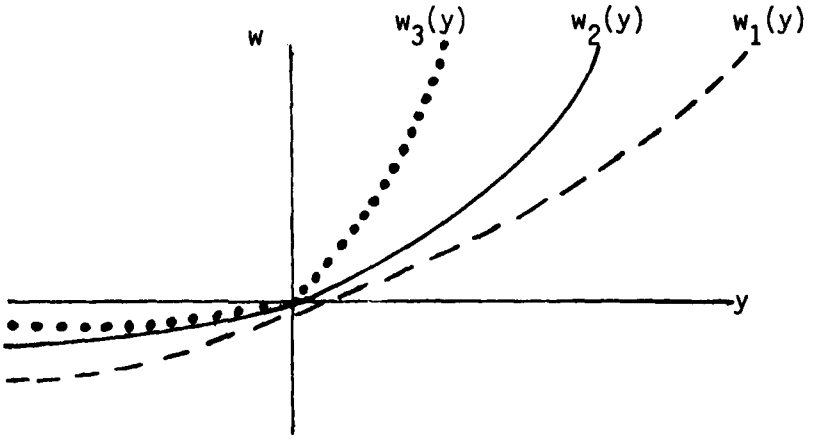


Figure 2.2 - A sequence of weighting functions

A family of weighting functions like the one in Figure 2.2 is constructed using a standard result from convexity theory (see e.g., [12]): if w(y) is any function satisfying Definition 2.1, and $\{p_i\}$

is any monotonically increasing sequence of positive real numbers, then the functions

$$w_i(y) = \frac{w(p_i \cdot y)}{p_i} \qquad i = 1, 2, \ldots \tag{2.5}$$

have monotonically increasing values on each nonzero y (as well as each trivially satisfying Definition 2.1). This is proven in somewhat more generality in Lemma 2.2.

Lemma 2.2 Let $w: R \to R$ be continuously differentiable and strictly convex, and assume $w(0) = 0$. Let $p_+ > p > 0$. Then for any $y \in R$,

$$\frac{w(p_+ \cdot y)}{p_+} \geq \frac{w(p \cdot y)}{p}$$

with equality if and only if $y = 0$.

Proof: If $y = 0$ the lemma is trivially true. Now suppose $y \neq 0$. Since w is continuously differentiable and strictly convex,

$$w(z) - w(x) > w'(x)(z - x) \tag{2.6}$$

for any $x, z \in R$, $x \neq z$. Substituting $x = p \cdot y$, $z = p_+ \cdot y$ into (2.6) gives

$$w(p_+ y) - w(py) > w'(py) \cdot (p_+ - p)\, y. \tag{2.7}$$

Also, substituting $x = py$, $z = 0$ into (2.6) and using $w(0) = 0$ gives

$$-w(py) > -w'(py) \cdot py$$

or

$$w'(py) \cdot y \geq \frac{w(py)}{p} \tag{2.8}$$

Combining (2.7) and (2.8)

$$w(p_+ y) - w(py) > \frac{w(py)}{p}(p_+ - p) = \frac{w(py)p_+}{p} - w(py)$$

or equivalently

$$\frac{w(p_+ y)}{p_+} > \frac{w(py)}{p}.$$

By solving a sequence of problems of form (2.4), with the weighting functions constructed according to (2.5), we have completed the development of our class of algorithms for solving feasibility problem (1.2II). It is given below. The parameter p is called by that letter because it resembles the penalty parameter found in various mathematical programming algorithms. D, $c_i(x)$ and $\ell_j(x)$ are defined as in problem (1.2), and we assume that the linear constraints have been found to be feasible in problem (1.2), part I.

Algorithm 2.3 - - A weighting function-penalty algorithm for determining feasibility of a set of nonlinear and linear inequality constraints (problem (1.2II)).

1. $p := 0$
2. solve $\min_{x \in D} \phi(x,p) = \frac{1}{p} \sum_{i=1}^{m} w(p \cdot c_i(x))$
   subject to $\ell_j(x) \leq 0 \quad j = 1, \ldots, k$
   say the minimum occurs at $x_*(p)$
3. if $\phi(x_*(p),p) > 0$, stop (constraints are infeasible)
   else if $x_*(p)$ feasible, stop (feasible point found)
   else increase p and return to step 2

Note that Algorithm 2.3 starts with $p = 0$. Since $w(0) = 0$, $\phi(x,0) = 0/0$, but L'Hôpital's rule shows that

$$\phi(x,0) = w'(0) \cdot \sum_{i=1}^{m} c_i(x) \tag{2.9}$$

where $w'(0)$ is some nonzero constant. So (2.9) is the first objective function used in step 2 of Algorithm 2.3. There are two advantages to starting with this objective function. First, the first linearly constrained optimization problem to be solved becomes a reasonably nice one, and yet its solution often establishes feasibility or

infeasibility in practice. Second, when m = 1, Algorithm 2.3 reduces to the algorithm proposed at the beginning of this section for solving (1.2II) with one nonlinear constraint.

The most important fact about Algorithm 2.3 is that it will now be shown in Section 3 that it is guaranteed to produce a feasible x, or establish infeasibility of the nonlinear constraints as soon as p reaches or exceeds some finite threshhold. (There is one exceptional case.) Thus, while one might fear that Algorithm 2.3 would only give the answer to the original feasibility problem (1.2II) in the limit as $p \to \infty$, it in fact gives the answer for finite p, and this is the key reason why it is an effective computational algorithm. We therefore feel that our class of algorithms is much more similar in its penalty characteristics to the "augmented Lagrangian" or "method of multipliers" methods for constrained optimization (see e.g., [10]), which are also guaranteed to terminate satisfactorily for finite p, than to the classical penalty or barrier algorithms for constrained optimization (see e.g., [5 , 9]) which only produce the correct answer in the limit as $p \to \infty$. It will be seen in Section 5 that typically few (1 to 3) values of p are required in practice.

3. Termination properties of the new class of algorithms

In this section, we establish the key property of the class of algorithms introduced in Section 2 (Algorithm 2.3) for determining feasibility of a set of nonlinear inequality constraints: that in almost all cases, the algorithm is guaranteed to either produce a feasible point, or establish infeasibility, as soon as the parameter p reaches or exceeds some finite value. This is proven in Theorems 3.4 - 3.6. Before this we prove two fundamental properties of our class of algorithms. We conclude the section by discussing the termination criteria which are used in the computer implementation of our algorithm.

For the purposes of this and the next section, there is no reason to make explicit mention of the linear constraints $\ell_j(x)$, $j = 1, \ldots, k$. Rather, we will assume that the domain D has been restricted, if necessary, to exclude any point x where some $\ell_j(x) > 0$. This simplifies our notation, and allows us to use the definitions of feasibility and infeasibility from Definition 1.1 without making any modification due to the linear constraints. Using this convention, an iteration of Algorithm 2.3 consists simply of minimizing $\phi(x,p)$ over $x \in D$. We will again define the point when this minimum occurs as $x_*(p)$, and for brevity will refer to the minimum value $\phi(x_*(p),p)$ as $\phi_*(p)$. This notation is summarized below.

Definition 3.1 Let $D \in R^n$, $c_i : D \to R$, $i = 1, \ldots, m$, and let $w : R \to R$. For $p > 0$, the composite weighting function $\phi(x,p)$ is defined by

$$\phi(x,p) = \frac{1}{p} \sum_{i=1}^{m} w(p \cdot c_i(x)).$$

For $p = 0$, $\phi(x,p)$ is defined by

$$\phi(x,0) = \lim_{p \to 0} \phi(x,p) = w'(0) \sum_{i=1}^{m} c_i(x).$$

For $p \geq 0$, we define

$$x_*(p) = \min_{x \in D} \phi(x,p)$$

and

$$\phi_*(p) = \phi(x_*(p),p).$$

In Lemma 3.2 we prove a property of our class of algorithms which was informally indicated in Section 2: that if $\{c_i\}$ is feasible, then $\phi_*(p)$ is nonpositive for all $p \geq 0$. This leads directly to the criterion which is used to establish infeasibility in our class of algorithms: if $\phi_*(p)$ is positive for any $p \geq 0$, then $\{c_i\}$ must be infeasible.

Note that the lemmas and theorems of this section assume only those properties of $w(y)$ which they actually use. However, taken together they require all the properties from Definition 2.1 except for $w''(y)$ continuous. This is required only by the linearly constrained optimization algorithm. Recall also that while in theory our results are true over the entire domain $x \in D$, in practice they may only be valid over whatever domain $\overline{D} \subseteq D$ the linearly constrained optimization routine can in fact find the minima of $\phi(x,p)$. That is, if the algorithm finds a feasible point the problem is of course feasible, but a claim of infeasibility may only be valid over the domain on which we are actually able to solve the minimization problems.

Lemma 3.2 Let $w : R \to R$ satisfy $w(0) = 0$ and $w(y) < 0$ for all $y < 0$, and let $c_i(x)$, $\phi(x,p)$, $x_*(p)$, $\phi_*(p)$ be defined by Definition 3.1. If $\{c_i\}$ is feasible, then for any $p \geq 0$, $\phi_*(p) \leq 0$. If $\{c_i\}$ is

strictly feasible, then for any $p \geq 0$, $\phi_*(p) < 0$.

Proof: If $\{c_i\}$ is feasible, then there exists some $\bar{x}$ such that $c_i(\bar{x}) \leq 0$, $i = 1, \ldots, m$. Thus, $\phi(\bar{x},p) \leq 0$ for any $p \geq 0$, and since by the definition of $\phi_*(p)$, $\phi_*(p) \leq \phi(\bar{x},p)$, we have $\phi_*(p) \leq 0$. The strictly feasible case is proven identically.

A question which might arise following Lemma 3.2 is: if $\phi_*(p) > 0$ for some fixed $p \geq 0$, is it possible that for some $p_+ > p$, $\phi_*(p_+) \leq 0$? If this were possible, then our test for infeasibility would be rather chancy, as we might "skip by" the choice p which would establish infeasibility, and use instead the choice p_+ which doesn't. Luckily this is not possible, because it is an easy consequence of the relation of successive weighting functions (Lemma 2.2) that $\phi_*(p)$ increases monotonically with p. (The one exception is that $\phi_*(p)$ can remain zero for two successive values of p, in which case the second optimal point, $x_*(p_+)$, must be feasible). This is proven in Lemma 3.3.

Lemma 3.3 Let $w : R \to R$ satisfy the assumptions of Lemma 2.2, and let $c_i(x)$, $\phi(x,p)$, $x_*(p)$, $\phi_*(p)$ be defined by Definition 3.1. Let $p_+ > p > 0$. Then

$$\phi_*(p_+) \geq \phi_*(p)$$

with equality only if $c_i(x_*(p_+)) = 0$, $i = 1, \ldots, m$.

Proof: From Lemma 2.3

$$\frac{w(p_+ \cdot c_i(x_*(p_+)))}{p_+} \geq \frac{w(p \cdot c_i(x_*(p_+)))}{p} \qquad (3.1)$$

$i = 1, \ldots, m$, with equality if and only if $c_i(x_*(p_+)) = 0$ for all i. Summing (3.1) for i going from 1 to m yields

$$\phi_*(p_+) \geq \phi(x_*(p_+),p) \qquad (3.2)$$

with equality if and only if $c_i(x_*(p_+)) = 0$, $i = 1, \ldots, m$.
Also, by the definition of $\phi_*(p)$

$$\phi(x_*(p_+),p) \geq \phi_*(p) \tag{3.3}$$

Combining (3.2) and (3.3) completes the proof.

Lemmas 3.2 and 3.3 give some insight into why Algorithm 2.3 will eventually solve the feasibility problem (1.2II). Since the minimum value of $\phi(x,p)$ is a monotonically increasing function of p, and since if $\phi_*(p)$ is positive for any p then the constraints are infeasible, we see that a kind of "squeeze play" is going on: as p increases, either $\phi_*(p)$ will become positive, or if the constraints are feasible, $\phi_*(p)$ will have to stay nonpositive. In the latter case, the weighting function will force $x_*(p)$ to eventually become feasible, because it will cause any infeasible point $\bar{x}$ to eventually have $\phi(\bar{x},p) > 0$.

This argument is the essense of our proofs that Algorithm 2.3 solves the feasibility problem (1.2II). The remaining important fact is that if $\{c_i\}$ is <u>strictly</u> feasible or <u>strictly</u> infeasible, Algorithm 2.3 will terminate satisfactorily as soon as p reaches or exeeds some finite value. This is proven in Theorems 3.4 and 3.5. In Theorem 3.6 we prove a slightly weaker result in the case when $\{c_i\}$ is exactly feasible (feasible but not strictly feasible). Recall that these three cases cover all possibilities, as the case of $\{c_i\}$ being infeasible but not strictly infeasible is impossible as long as D is closed and bounded.

Theorem 3.4 proves the finite termination of Algorithm 2.3 if $\{c_i\}$ is strictly infeasible. The proof basically proceeds by showing that due to the nature of the weighting function, eventually $\phi(\bar{x},p) > 0$ for any strictly infeasible point $\bar{x}$.

Theorem 3.4. Let $w: R \to R$ be a monotonically increasing function of y, and assume $\lim_{y\to-\infty} w(y) = \ell$ for some finite $\ell < 0$, $\lim_{y\to+\infty} w(y) = +\infty$. Let $c_i(x)$, $\phi(x,p)$, $x_*(p)$, $\phi_*(p)$ be defined by Definition 3.1. If $\{c_i\}$ is strictly infeasible, then there exists $p_1 \geq 0$ such that for any $p \geq p_1$, $\phi_*(p) > 0$.

Proof: Since $\{c_i\}$ is strictly infeasible, there exists $\delta_1 > 0$ such that for all $x \in D$, $c_i(x) > \delta_1$ for at least one i. Let y_1 be chosen such that

$$w(y_1) \geq -(m-1)\ell \tag{3.4}$$

and define $p_1 = y_1 / \delta_1$. We show that for any $p \geq p_1$, $\phi_*(p) > 0$.

Since $\{c_i\}$ is not within δ_1 of feasibility, by the monotonicity of w we have

$$\phi_*(p) > \frac{(m-1)\ell + w(p\delta_1)}{p} \tag{3.5}$$

From the definition of p_1 and y_1 and the monotonicity of w,

$$w(p\delta_1) \geq w(p_1\delta_1) = w(y_1) \geq -(m-1)\ell \tag{3.6}$$

Combining (3.5) and (3.6) completes the proof.

Theorem 3.5 proves the finite termination of Algorithm 2.3 if $\{c_i\}$ is strictly feasible. The proof basically procedes by showing that due to the nature of the weighting function, eventually $\phi(\bar{x},p) > \phi(\hat{x},p)$ for any infeasible point $\bar{x}$ and any strictly feasible point $\hat{x}$.

Theorem 3.5. Let $w: R \to R$ be a monotonically increasing function of y with $w(0) = 0$, and assume $\lim_{y\to-\infty} w(y) = \ell$ for some finite $\ell < 0$. Let $c_i(x)$, $\phi(x,p)$, $x_*(p)$, $\phi_*(p)$ be defined by Definition 3.1. If $\{c_i\}$ is strictly feasible, then there exists $p_2 \geq 0$ such that for any $p \geq p_2$, $x_*(p)$ is feasible.

Proof: Since $\{c_i\}$ is strictly feasible, there exists $\varepsilon_2 < 0$, $x_2 \in R^n$ such that $c_i(x_2) \leq \varepsilon_2$, $i = 1, \ldots, m$. Let y_2 be chosen such that

$$w(y_2) \leq \frac{(m-1)\ell}{m} \tag{3.7}$$

and define $p_2 = y_2/\varepsilon_2$. We show that for any $p \geq p_2$, $x_*(p)$ is feasible.

Suppose that for some $p \geq p_2$, $x_*(p)$ is infeasible. Then since at least one constraint is infeasible at $x_*(p)$ and $w(y) > 0$ for all $y > 0$,

$$\phi_*(p) > \frac{(m-1)\ell}{p} . \tag{3.8}$$

Also, from the definition of x_2 and the monotonicity of w,

$$\phi(x_2,p) \leq \frac{m(w(p\,\varepsilon_2))}{p} \tag{3.9}$$

From the definitions of p_2 and y_2 and the monotonicity of w,

$$w(p\varepsilon_2) \leq w(p_2\varepsilon_2) = w(y_2) \leq \frac{(m-1)\ell}{m} \tag{3.10}$$

so that from (3.9) and (3.10)

$$\phi(x_2.p) \leq \frac{(m-1)\ell}{p} \tag{3.11}$$

From (3.8) and (3.11), $\phi(x_2,p) < \phi_*(p)$ which is a contradiction and completes the proof.

The remaining case is when $\{c_i\}$ is exactly feasible. While this case may not be very meaningful in finite precision arithmetic, results indicating how Algorithm 2.3 performs on it in theory should indicate how it will perform on "close to exactly feasible" problems in practice. In Theorem 3.6 we show that if $\{c_i\}$ is exactly feasible, then given any $\delta > 0$, Algorithm 2.3 is guaranteed to produce an $x_*(p)$ that is within δ of feasibility as soon as p reaches or exceeds some finite limit. We comment on the optimality of this result after the proof. The proof of Theorem 3.6 is closely related to that of Theorem 3.4.

Theorem 3.6. Let $w : R \to R$ satisfy the assumption of Theorem 3.5, and assume in addition that $\lim_{y \to +\infty} w(y) = +\infty$. Let $c_i(x)$, $\phi(x,p)$ $x_*(p)$, $\phi_*(p)$ be defined by Definition 3.1. If $\{c_i\}$ is feasible, then given any $\delta_3 > 0$, there exists $p_3 \geq 0$ such that for any $p \geq p_3$, $x_*(p)$ is within δ_3 of feasibility.

Proof: Let y_1 be defined by (3.4) and define $p_3 = y_1/\delta_3$. We show that for any $p \geq p_3$, $x_*(p)$ is within δ_3 of feasibility.

Suppose that for some $p \geq p_3$, $x_*(p)$ is not within δ_3 of feasibility. Then $c_i(x_*(p)) > \delta_3$ for at least one i, and so using the monotonicity of w,

$$\phi_*(p) > \frac{(m-1)\ell + w(p \cdot \delta_3)}{p} \tag{3.12}$$

From the definitions of p_3 and y_1 and the monotonicity of w,

$$w(p \cdot \delta_3) \geq w(p_3 \delta_3) = w(y_1) \geq -(m-1)\ell \tag{3.13}$$

so that from (3.12) and (3.13)

$$\phi_*(p) > 0. \tag{3.14}$$

On the other hand, we have from Lemma 3.2 that

$$\phi_*(p) \leq 0$$

which contradicts (3.14) and completes the proof.

Theorem 3.6 implies that the limit of the points generated by Algorithm 2.3,

$$x_* = \lim_{p \to \infty} x_*(p)$$

will be feasible for exactly feasible problems. This is a weaker result than for strictly feasible problems. Thus a reasonable question to ask is whether our result for strictly feasible problems can be extended to exactly feasible problems, i.e., whether there will exist any finite p such that $x_*(p)$ is feasible in this case.

Example 3.7 shows that the answer is no for some exactly feasible problems, but yes for some others. Thus Thecrem 3.6 is the best result we can prove about our algorithm's performance on exactly feasible problems in general, but the algorithm may do better on some exactly feasible problems.

Example 3.7. Assume w satisfies Definition 2.1, let $c_i(x)$, $\phi(x,p)$, $x_*(p)$, $\phi_*(p)$ be defined by Definition 3.1, and assume the set of constraints $\{c_i\}$ is exactly feasible. Then it is possible that $x_*(p)$ is infeasible for all $p > 0$; it is also possible that $x_*(p)$ is feasible for all $p > 0$.

This is shown via 2 examples. Consider first

$$c_1(x) = 1 - x^3$$
$$c_2(x) = x^2 - 1$$

which has its sole feasible point at $x = 1$. For $p = 0$, $x_*(p)$ is undefined. For $p > 0$, $x_*(p)$ must satisfy

$$\frac{d}{dx}\phi(x,p) = 0 = -3x^2w'(p-px^3) + 2xw'(px^2-p).$$

It is clear that $x = 1$ is never a solution to this equation, as this would imply $-w^1(0) = 0$ which contradicts the monotonicity of w. Thus $x_*(p)$ must be infeasible for all $p > 0$. Incidentally, for sufficiently large p,

$$x_*(p) \simeq 1 - \left[\frac{w'(0)}{13pw''(0) - 3w'(0)}\right]$$

so that $\lim_{p\to\infty} x_*(p) = 1$.

Now consider

$$c_1(x) = 1 - x^3$$
$$c_2(x) = x^3 - 1$$

which also has its sole feasible point at $x = 1$. For $p = 0$, $x_*(p)$ is again undefined. For $p > 0$

$$\begin{aligned} \phi(x,p) &= \frac{w(p \cdot c_1(x)) + w(p \cdot c_2(x))}{p} \\ &= \frac{w(p \cdot c_1(x)) + w(-p \cdot c_1(x))}{p} . \end{aligned} \tag{3.15}$$

From $w(0) = 0$ and w strictly convex, we have

$$w(a) + w(-a) \geq 0$$

for any $a \in R$, with equality if and only if $a = 0$. Thus for any $p > 0$, the unique minimum to (3.15) occurs when $p \cdot c_1(x) = 0$, i.e., when $c_1(x) = c_2(x) = 0$. Thus $x_*(p) = 1$ for all $p > 0$.

Theorems 3.4 - 3.6 show that Algorithm 2.3 will solve the feasibility problem (1.2II) correctly. However, in the exactly feasible case this may only be true in the limit as $p \to \infty$; correspondingly, if $\{c_i\}$ is strictly feasible only for very small (in absolute value) negative ε, or strictly infeasible only for very small positive δ, the correct solution may require p to be very large. In practice, one probably wants to limit how large p can get in Algorithm 2.3. Therefore, in our computer implementation of this algorithm, we have changed the termination criteria to:

$$\begin{aligned} &\text{if } \phi_*(p) > 0, \underline{\text{stop}} \text{ (constraints are infeasible)} \\ &\text{if } x_*(p) \text{ is feasible or within } \delta \text{ of feasibility, } \underline{\text{stop}} \\ &\quad \text{(constraints are considered feasible)} \end{aligned} \tag{3.16}$$

where δ is a small positive number. It is shown in Theorem 3.8 that Algorithm 2.3 with stopping criteria (3.16) is guaranteed to terminate as soon as p reaches or exceeds $\bar{p} = y_1(m+1)/\delta$, y_1 given by equation (3.4). The algorithm will be guaranteed to produce the correct result

to feasibility problem (1.2II) unless $\{c_i\}$ is infeasible but within δ of feasibility. If δ is a small number, then for most applications this uncertainty will be acceptable, and in some applications we have seen this "benefit of the doubt" is even desirable. A possible disadvantage is that some applications really need a feasible point if one exists, and an $x_*(p)$ which is infeasible but within δ of feasibility will not do. In the unlikely event that this occurs, one can continue to iterate Algorithm 2.3 in the hope that it will find a feasible point.

Theorem 3.8. Let $w : R \to R$ satisfy the assumptions of Theorem 3.6, and let $c_i(x)$, $\phi(x,p)$, $x_*(p)$, $\phi_*(p)$ be defined by Definition 3.1. Then given any $\delta_4 > 0$, there exists $p_4 \geq 0$ such that for any $p \geq p_4$, either $\phi_*(p) > 0$ or $x_*(p)$ is within δ_4 of feasibility.

Proof: Let y_1 be defined by (3.4) and define $\gamma = \delta_4 / (m+1)$, $p_4 = y_1 / \gamma$. Our proof is divided into two cases. We show that if $\{c_i\}$ is not within γ of feasibility, then $\phi_*(p) > 0$ for any $p \geq p_4$, and that if $\{c_i\}$ is either feasible or within γ of feasibility, that $x_*(p)$ is within δ_4 of feasibility for any $p \geq p_4$.

If $\{c_i\}$ is not within γ of feasibility, then it is shown in Theorem 3.4 that $\phi_*(p) > 0$ for any $p \geq p_4$. Now consider the other possibility, that $\{c_i\}$ is feasible or within γ of feasibility. Then there exists x_4 such that $c_i(x_4) \leq \gamma$, $i = 1, \ldots, m$, so that for any $p > 0$,

$$\phi(x_4,p) \leq \frac{m \cdot w(p\gamma)}{p} . \tag{3.17}$$

Now suppose that for some $p \geq p_4$, $x_*(p)$ is not within δ_4 of feasibility. Then by the monotonicity of w,

$$\phi_*(p) > \frac{(m-1)\ell + w(p\delta_4)}{p} \tag{3.18}$$

Since w is strictly convex and $w(0) = 0$,

$$w(p\delta_4) > (m+1)\, w\left(\frac{p\delta_4}{m+1}\right) = (m+1)\, w(p\gamma) \tag{3.19}$$

so that combining (3.18) and (3.19)

$$\phi_*(p) > \frac{[(m-1)\ell + w(p\gamma)] + m \cdot w(p\gamma)}{p} . \tag{3.20}$$

Also by the monotonicity of w and the definitions of p_4 and y_1,

$$w(p\gamma) \geq w(p_4\gamma) = w(y_1) \geq -(m+1)\ell \tag{3.21}$$

so that combining (3.20) and (3.21),

$$\phi_*(p) > \frac{m \cdot w(p\gamma)}{p} . \tag{3.22}$$

However, from (3.17) and (3.22), $\phi(x_4,p) < \phi_*(p)$ which is a contradiction and completes the proof.

A logical question to ask now is: since Algorithm 2.3 using stopping criteria (3.16) is guaranteed to terminate for $\bar{p} = y_1(m+1)/\delta$, where y_1 is readily calculated given w and m, why not just solve the linearly constrained optimization problem (2.4) with $p = \bar{p}$, instead of executing Algorithm 2.3 which solves (2.4) for a sequence of p's? There are two reasons. First, since $\phi(x,p)$ becomes more poorly behaved as p increases, it may be easier to solve (2.4) for a sequence of p's terminating with $p = \bar{p}$, using the solution to each as the starting guess to the next, than to just solve the problem with $p = \bar{p}$. In practice, linearly constrained optimization problems after the first in Algorithm 2.3 require few iterations. Second, it is very possible that Algorithm 2.3 will terminate for some $p < \bar{p}$.

Another important question which we have not yet addressed is whether our techniques can be used to obtain a strictly feasible

point in cases when $\{c_i\}$ is strictly feasible. The answer is yes, but how strictly feasible turns out to depend on the weighting function w(y). This is one aspect of the choice of the weighting function which is discussed in Section 4.

4. Choice of the weighting function

In Algorithm 2.3 we have proposed a class of algorithms for solving the feasibility problem (1.2II). This class involves a weighting function w(y) which must obey Definition 2.1, with a specific algorithm involving a specific choice of the weighting function. In Section 3 we have proven that any algorithm from this class will successfully solve the feasibility problem. In this section, we discuss the choice of the weighting function w(y). First, we show that a variety of functions exist satisfying Definition 2.1. Then we examine which of these is likely to cause quicker termination of Algorithm 2.3. Finally, we analyze (in Theorem 4.1, Corollary 4.2 and Example 4.3) the influence of the weighting function on the ability of Algorithm 2.3 to find an interior point to a set of strictly feasible constraints. From these considerations, a preference emerges for the weighting function to use in practice.

The simplest (to write) function which satisfies Definition 2.1 is

$$w_1(y) = e^y - 1.$$

Indeed, $w_1(y)$ and its related multiples and powers are the only infinitely differentiable functions we know of which satisfy Definition 2.1. However, many twice continuously differentiable functions satisfying Definition 2.1 can be constructed by "splicing together" two functions. For example, one can let $w(y) = \frac{y}{1-y}$ for $y \le \alpha < 1$, and then join it to a quadratic for $y \ge \alpha$ in a way which makes w(y) twice continuously differentiable. This is done below and drawn in Figure 4.1.

$$w_2(y) = \begin{cases} \dfrac{y}{1-y} & y \le \alpha < 1 \\ \dfrac{(y-\alpha)^2}{(1-\alpha)^3} + \dfrac{(y-\alpha)}{(1-\alpha)^2} + \dfrac{\alpha}{1-\alpha} & y \ge \alpha \end{cases}$$

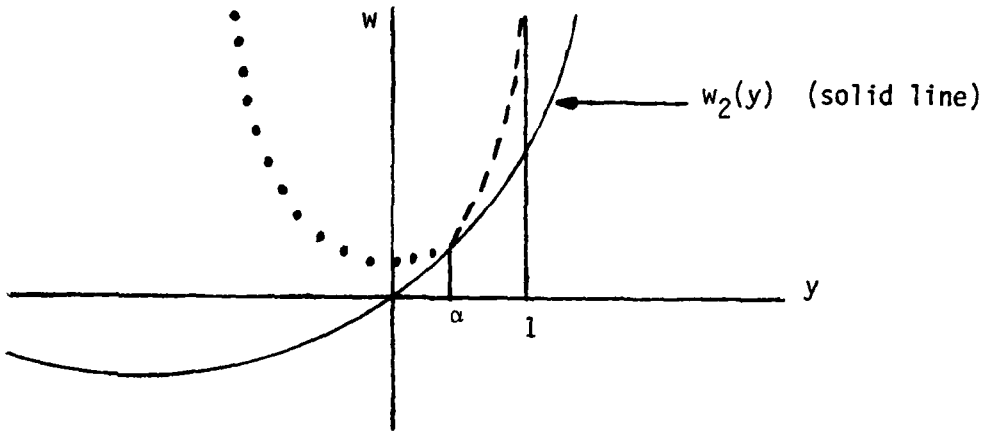


Figure 4.1

In the same manner one can form a weighting function by splicing together $(e^y - 1)$ and a quadratic at a point $\beta > 0$. This is done in $w_3(y)$ and drawn in Figure 4.2. Function $w_3(y)$ may be preferable to $w_1(y)$ in practice because it grows less quickly for large y and thus diminishes the chances of overflow or badly behaved optimization problems in Algorithm 2.3.

$$w_3(y) = \begin{cases} e^y - 1 & y \le \beta > 0 \\ \frac{e^\beta}{2}(y-\beta)^2 + e^\beta(y-\beta) + e^\beta - 1 & y \ge \beta \end{cases}$$

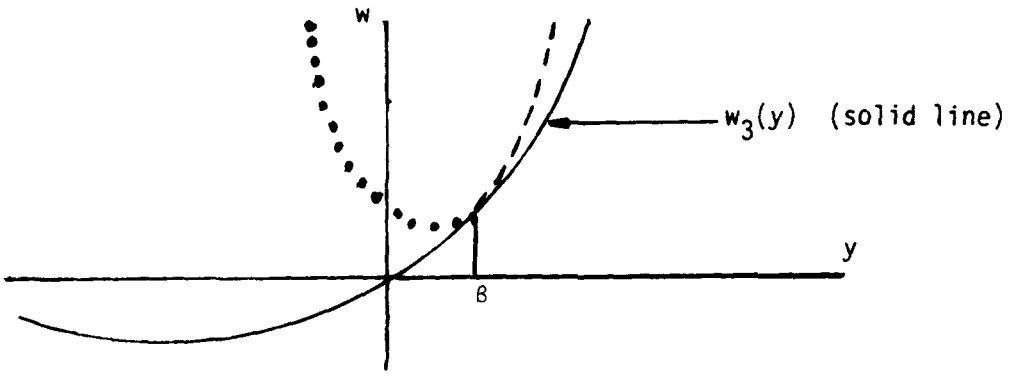


Figure 4.2

Certainly other functions satisfying Definition 2.1 are possible, but for our purposes it will suffice to consider these three. In particular, much of our analysis turns out to concern only the portion of $w(y)$ for $y \le 0$, and here the functions $w_1(y) = e^y - 1$ and $w_2(y) = y/(1-y)$ seem to be the two important possibilities.

One way to compare the weighting functions is to examine the upper bounds produced by Theorems 3.4 - 3.6 and 3.8 on how large p must be to guarantee termination of Algorithm 2.3. These bounds are all functions of the weighting function, and we are interested in which weighting function leads to the lowest bounds. The easiest case to examine is the upper bound on p when $\{c_i\}$ is strictly ε feasible for some $\varepsilon \le 0$. Theorem 3.5 tells us that Algorithm 2.3 is guaranteed to terminate for $p \ge y_2/\varepsilon$, where y_2 satisfies

$$w(y_2) = \frac{(m-1)\ell}{m} .$$

If $w(y) = e^y - 1$ for $y < 0$, it is easily calculated that $\ell = -1$ and

$$y_2 = -\ell n\, m.$$

If $w(y) = y/(1-y)$ for $y < 0$, again $\ell = -1$ and

$$y_2 = -(m-1).$$

Since $\ell n\, m < m-1$ for all $m \geq 2$ (and recalling $\varepsilon < 0$) we see that the upper bound using $e^y - 1$ is less than the upper bound if one uses $y/(1-y)$. Note that this analysis only involves the portion of $w(y)$ for $y < 0$. Thus using $w(y) = e^y - 1$ for $y < 0$ may be preferable to using $w(y) = y/(1-y)$ for $y < 0$ on strictly feasible problems.

Now consider the bound given by Theorem 3.4 in the case when $\{c_i\}$ is strictly δ infeasible for some $\delta > 0$. The theorem says that Algorithm 2.3 is guaranteed to terminate for $p \geq y_1 / \delta$, where y_1 satisfies

$$w(y_1) = -(m-1)\,\ell.$$

If $w(y) = e^y - 1$ then

$$y_1 = \ell n\, m.$$

If $w(y) = w_2(y)$ then

$$y_1 = \begin{cases} \dfrac{m-1}{m} & \alpha \geq \dfrac{m-1}{m} \\[2ex] \dfrac{3\alpha - 1}{2} + \dfrac{1-\alpha}{2}\sqrt{4m(1-\alpha) - 3} & \alpha \leq \dfrac{m-1}{m} \end{cases}$$

Thus the situation here is less clear cut than the above. If $\alpha > (m-1)/m$, then the upper bound for $w_2(y)$ clearly is less than the upper bound for $w_1(y)$. If $\alpha < (m-1)/m$ then the bound for $w_2(y)$ may exceed the bound for $w_1(y)$. The bound for $w_3(y)$ is always greater than or equal to the bound for $w_1(y)$. Note that this analysis only involves $w(y)$ for $y > 0$.

However, the selection of the weighting function for $y > 0$ is complicated by the presense of two conflicting considerations: the more quickly $w(y)$ increases, the lower will be the upper bound on p required by Theorem 3.4, but the larger will be the chance for overflow or bad behavior in the objective function $\phi(x,p)$ if p does become large. So while we could always set $\alpha > (m-1)/m$ and thus have a smaller y_1 for $w_2(y)$ than for $w_1(y)$ or $w_3(y)$, we might not want to do this because it might make $w(y)$ too steep for $y > 0$. Similarly, Theorems 3.6 and 3.8 show that if we stop our algorithm when $x_*(p)$ is within δ of feasibility, then the upper bound on p is again proportional to y_1, but the same conflicting considerations cloud the selection of $w(y)$. Thus we believe that only experimentation can determine which weighting function is preferable for $y > 0$.

The other important regard in which the selection of the weighting function can influence the performance of Algorithm 2.3 is the ability of the algorithm to find strictly feasible points when $\{c_i\}$ is strictly feasible. While Algorithm 2.3 as written will stop as soon as a feasible point is found, some applications would like a point which is as feasible as possible. Therefore, we might be curious whether, if we continue to run Algorithm 2.3 after it has found a feasible point to a strictly ε feasible problem, it will eventually find an $x_*(p)$ which is "almost as feasible as possible," i.e., strictly $\hat{\varepsilon}$ feasible for $\hat{\varepsilon}$ greater than but arbitrarily close to ε. Theorem 4.1 enables us to answer this question, by giving a necessary condition for Algorithm 2.3 to eventually produce a strictly $\hat{\varepsilon}$ feasible $x_*(p)$ on a strictly ε feasible problem for a given $\hat{\varepsilon}$ in $(\varepsilon,0)$. This condition involves $w(y)$ only for $y < 0$. Corollary 4.2 then shows

that $w(y) = e^y - 1$ satisfies this condition for any $\hat{\varepsilon}$ in $(\varepsilon,0)$. Thus Algorithm 2.3 with $w(y) = e^y - 1$ for $y < 0$ will produce points which are "almost as feasible as possible." However, $w(y) = y/(1-y)$ does not satisfy the condition of Theorem 4.1 for $\hat{\varepsilon} \leq \varepsilon/m$, and indeed it is shown in Example 4.3 that continuing Algorithm 4.3 with $w(y) = y/(1-y)$ for $y < 0$ may fail to produce points which are "almost as feasible as possible."

Theorem 4.1. Let w obey the assumptions of Theorem 3.5 and define $\hat{w}(y) = w(y) - \ell$. Let $c_i(x)$, $\phi(x,p)$, $x_*(p)$, $\phi_*(p)$ be defined by Definition 3.1. Let $\varepsilon < \hat{\varepsilon} < 0$, and assume $\{c_i(x)\}$ is strictly ε feasible. Suppose there exists $p_5 \geq 0$ such that for all $p \geq p_5$,

$$m \cdot \hat{w}(p\varepsilon) \leq \hat{w}(p\hat{\varepsilon}). \tag{4.1}$$

Then for any $p \geq p_5$, $x_*(p)$ is strictly $\hat{\varepsilon}$ feasible.

Proof: Suppose there exists p_5 satisfying the conditions of the theorem, and that for some $p \geq p_5$, $x_*(p)$ is not $\hat{\varepsilon}$ feasible. Then at least one constraint has value greater than $\hat{\varepsilon}$ at $x_*(p)$, so that

$$\phi_*(p) > \frac{(m-1)\ell + w(p\hat{\varepsilon})}{p} = \frac{m \cdot \ell + \hat{w}(p\hat{\varepsilon})}{p} \tag{4.2}$$

However, since $\{c_i\}$ is strictly ε feasible, there exists $x_5 \in R^n$ such that $c_i(x_5) \leq \varepsilon$, $i = 1, \ldots, m$. Thus from the definition of $x_*(p)$ and the monotonicity of w,

$$\phi_*(p) \leq \frac{mw(p\varepsilon)}{p} = \frac{m \cdot \ell + m\hat{w}(p\varepsilon)}{p} \tag{4.3}$$

Equations (4.2) and (4.3) imply that

$$\hat{w}(p\hat{\varepsilon}) < m \cdot \hat{w}(p\varepsilon)$$

which contradicts (4.1) and completes the proof.

Corollary 4.2. Let $w(y) = e^y - 1$ for $y < 0$, and let $c_i(x)$, $\phi(x,p)$,

$x_*(p)$ be defined by Definition 3.1. Let $\varepsilon < \hat{\varepsilon} < 0$ and assume that $\{c_i\}$ is strictly ε feasible. Then there exists $p_6 \geq 0$ such that for all $p \geq p_6$, $x_*(p)$ is strictly $\hat{\varepsilon}$ feasible.

Proof: From Theorem 4.1, it suffices to show that there exists p_6 such that for all $p \geq p_6$, (4.1) is true. For $w(y) = e^y - 1$, $\hat{w}(y) = e^y$, so (4.1) requires that

$$me^{p\varepsilon} \leq e^{p\hat{\varepsilon}}$$

or

$$m \leq e^{p(\hat{\varepsilon} - \varepsilon)}.$$

This clearly is true for any $p \geq \frac{\ell n\, m}{\hat{\varepsilon} - \varepsilon} \triangleq p_6$.

Now let us consider what Theorem 4.1 says about the weighting function $w(y) = y/(1-y)$, $y < 0$. In this case $\hat{w}(y) = 1/(1-y)$, and so (4.1) requires

$$\frac{m}{1 - p\varepsilon} \leq \frac{1}{1 - p\hat{\varepsilon}}$$

or

$$m-1 \leq p(m\hat{\varepsilon} - \varepsilon)$$

which is only possible if

$$\hat{\varepsilon} > \frac{\varepsilon}{m}.$$

Thus with this weighting function, Theorem 4.1 only guarantees that continuing to iterate Algorithm 2.3 on a strictly ε feasible problem will eventually produce an $x_*(p)$ which is strictly $\hat{\varepsilon}$ feasible for $\hat{\varepsilon} > \varepsilon/m$. Indeed, in Example 4.3 we give an example where Algorithm 2.3 does not produce any $x_*(p)$ which is strictly (-2.5) feasible on a problem which is strictly (-3) feasible. For simplicity we let the constraints be linear; more extreme examples are possible for

nonlinear constraints.

Example 4.3. Let $w(y) = \frac{y}{1-y}$ for $y < 0$, and let $c_i(x)$, $\phi(x,p)$, $x_*(p)$ be defined by Definition 3.1. Let $\varepsilon < 0$ and assume that $\{c_i\}$ is strictly ε feasible. Then there may exist some $\hat{\varepsilon}$ in $(\varepsilon,0)$ such that for all $p \geq 0$, $x_*(p)$ is not $\hat{\varepsilon}$ feasible.

As an example, consider

$$c_1(x) = -2-x$$
$$c_2(x) = -10+7x$$

(See Figure 4.3)

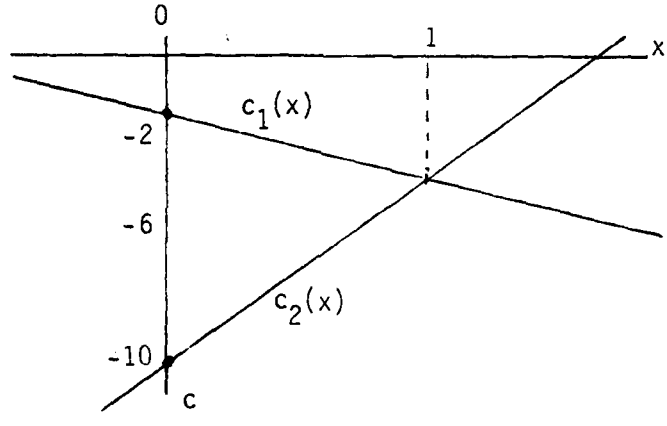


Figure 4.3

This set of constraints is clearly strictly ε feasible for $\varepsilon = -3$, as $c_1(1) = c_2(1) = -3$. However, it will be shown that $x_*(p) < \frac{1}{2}$ for all $p \geq 0$. Thus $x_*(p)$ is not ε feasible for any $\varepsilon \leq -2.5$. The proof of this is as follows.

For $p = 0$, $x_*(p)$ is undefined. For $p > 0$,

$$x_*(p) = h(p) = (7(1+6p) - \sqrt{49(1+6p)^2 + 21(3+4p-36p^2)}) / 21p$$

The term $h(p)$ is positive for $p > (2\sqrt{7}-1)/18 \approx 0.238$, and is monotonically increasing for $p \geq 0$ as

$$h'(p) = \frac{1}{3p^2}\left[-1 + \frac{7 + 24p}{\sqrt{(7+24p)^2 - (6+18p)^2}}\right]$$

which is clearly greater than zero for all $p \geq 0$. Finally

$$\lim_{p\to\infty} h(p) = 2 - \frac{4\sqrt{7}}{7} \simeq .488 \triangleq x_* .$$

Since $c_1(x_*) \simeq -2.488$, it is clear from Figure 4 that no $x_*(p)$ is strictly ε feasible for any $\varepsilon < -2.5$.

Thus with regard to finding interior points, the choice $w(y) = e^y - 1$ for $y < 0$ seems superior to $w(y) = y/(1-y)$. This concurs with our analysis earlier in this section. For $y > 0$ the issue is unsettled by theoretical analysis. Thus in practice one probably wants to experiment between $w_1(y)$, $w_3(y)$, and possibly a hybrid which combines $e^y - 1$ for $y < 0$ with a function like $w_2(y)$ for $y \geq 0$.

It should be mentioned that the class of weighting functions satisfying Definition 2.1 is not the broadest class of twice continuously differentiable functions for which the results of this paper are true. For example, I. Zang [13] has suggested to us that we consider weighting functions which obey Definition 2.1 except that they are linear in some intervals. An example (related to one in [14]) is given below and drawn in Figure 4.4.

$$w_4(y) = \begin{cases} \dfrac{-3\sigma}{16} & y \le -\sigma < 0 \\ \dfrac{-(y+\sigma)^4}{16\sigma^3} + \dfrac{(y+\sigma)^3}{4\sigma^2} - \dfrac{3\sigma}{16} & -\sigma \le y \le \sigma \\ y - \dfrac{3\sigma}{16} & y \ge \sigma \end{cases}$$

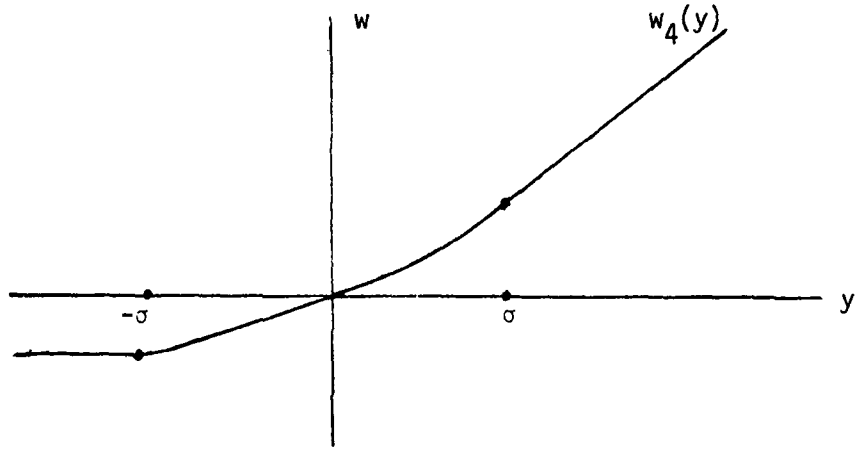


Figure 4.4

The class of functions which look like this, that is, obey Definition 2.1 except that they may be strictly convex only in some open neighborhood of y = 0 and convex elsewhere, is a class to which all the results of this paper are readily extendable. It may prove to be attractive computationally.

5. Computational results

In this section we discuss some important aspects of the computer implementation of our algorithm for solving feasibility problem (1.2II), and present some sample computational results. Due to the length of this paper, and because we are continuing to revise and experiment with the software for our algorithm, our treatment in this section is brief. A future paper will be specifically devoted to the implementation and test results of our algorithm.

In implementing Algorithm 2.3, three important aspects need to be considered:

1. the choice of the weighting function
2. the linearly constrained optimization algorithm, especially the strategy to attempt to find the global minimum
3. the strategy for increasing p.

The choice of the weighting function was discussed in Section 4. In our implementation so far we have used $w(y) = e^y - 1$, which is supported by our analysis there. We have not had any trouble with overflows or badly conditioned optimization problems. As is mentioned in Section 4, we also intend to experiment with the exponential-quadratic hybrid $w_3(y)$.

The linearly constrained optimization algorithm is the heart of our computer algorithm. Our requirements for this algorithm are different enough from the standard ones that we deemed it preferable to write our own code rather than modify an existing one. In the initial version of our algorithm, we have developed an algorithm similar to the ones proposed by Fletcher [6] and Goldfarb [8]. The algorithm

uses finite difference approximations to the first and second derivatives of $\phi(x,p)$ (i.e., of $c_i(x)$, $i = 1, \ldots, m$) because this is reasonable in the applications with which we are familiar. If the projected Newton direction is not a descent direction then it takes a step in the projected steepest descent direction, although clearly this aspect of the algorithm can be improved. Otherwise, it attempts to find a local minimum of $\phi(x,p)$ with respect to x, in a standard manner. Along the way, it attempts to obtain information indicating whether a lower value of $\phi(x,p)$ might exist elsewhere. Information is gathered from directions of negative curvature (if any are encountered) indicating regions of the domain in which a lower value might be found. Then, after a local minimum has been found, this information together with a sampling of points near and far from the local minimum is used to determine whether a restart of the algorithm in another region is likely to produce a lower value of $\phi(x,p)$. If so, the algorithm is restarted and this entire process is repeated (possibly more than once). Such a procedure is certainly ad-hoc and will not work on every problem, but it has proven successful on the majority of problems with multiple local minima that we have tried.

Our linearly constrained optimization algorithm also requires some special features due to the setting in which it is used. Certainly, if at any stage in the minimization of $\phi(x,p)$ the current iterate x_c is feasible, then there is no need to continue the minimization and so both the linearly constrained optimization algorithm and Algorithm 2.3 are terminated. Secondly, while the value of $\phi(x,p)$ is bounded below for any $p > 0$, it is possible that it has no finite minimum

point x_* if the domain D is not bounded. This is treated in the standard way by terminating the local minimization of $\phi(x,p)$ if successive iterates of x lead to insufficient decrease in $\phi(x,p)$. Finally, when $p = 0$ it is possible that $\phi(x,0) = \sum_{i=1}^{m} c_i(x)$ is unbounded below if D is unbounded, and that the algorithm may pursue an unbounded path without considering any feasible points. Two simple examples are

$$c_1(x) = x^2 - 1$$
$$c_2(x) = x^3$$

which is feasible and

$$c_1(x) = x^2 + 1$$
$$c_2(x) = x^3$$

which is infeasible. In both cases $\phi(x,0)$ decreases without limit as $x \to -\infty$, and the algorithm will pursue this path without locating a feasible point if the initial x is less than -1. Thus our linearly constrained optimization algorithm treats the case $p = 0$ specially, and in particular, once $\phi(x,0) < 0$, it detects situations like the above and terminates the minimization of $\phi(x,0)$ prematurely. Algorithm 2.3 then increases p and continues.

Our entire algorithm for linearly constrained optimization in the context of Algorithm 2.3 is currently undergoing major revision, and will be reported in more detail on in the future. Our initial algorithm has worked well enough for our implementation of Algorithm 2.3 to be quite successful.

The strategy for increasing p has been given considerable attention, although perhaps surprisingly Algorithm 2.3 does not seem too sensitive to it. At each iteration of Algorithm 2.3, we model the

function $\phi_*(p) = \phi(x_*(p),p)$ by a function $m(p)$, and then choose the next value p_+ such that $m(p_+) = 0$. In this way we attempt to "force the issue" by either producing a value of $\phi_*(p) > 0$, or a feasible $x_*(p)$. The model function $m(p)$ is designed to fit (5.1)

$$m(p_c) = \phi_+(p_c),\ m(p_{prev}) = \phi_*(p_{prev}),\ m'(p_c) = \frac{d}{dp}\phi(x_*(p),p)\Big|_{p=p_c}$$

where p_c and p_{prev} are the current and immediately previous values of p. The derivative in (5.1) is an easily obtainable approximation to $\phi_*'(p_c)$; in a number of tests taking finite difference approximations to the latter, we found the two derivatives to be almost the same. While one can fit (5.1) with a quadratic, it seems preferable to make use of the problem structure. We currently use

$$m(p) = \alpha w(p \cdot \beta) + \gamma$$

where α, β and γ are chosen to satisfy (5.1); a perhaps more attractive, but more difficult to obtain, possibility is

$$\hat{m}(p) = \frac{\hat{\alpha} w(p \cdot \hat{\beta})}{p} + \hat{\gamma}$$

If the predicted p_+ is not in the range $[2p_c, 10p_c]$ we use the appropriate extremum of this range as p_+. In the initial case ($p_c = 0$) the model is constructed somewhat differently.

In Tables 5.1 - 5.4 we present four indicative runs of our initial implementation of Algorithm 2.3. Mainly they illustrate that the algorithm works in practice about as is predicted by our theory. What is perhaps most striking in our tests so far is how few values of p seem to be required to obtain the correct answer. Table 5.1 and 5.2 illustrate the algorithm on feasible problems. Table 5.1 is a case where the global characteristics of our linearly constrained

optimization were required due to the shape of the tilted sine function. Tables 5.3 and 5.4 illustrate the algorithm on infeasible problems. Table 5.4 is interesting because the optimal point $x_*(p)$ remains the same for all p, but p must reach a positive threshhold value before $\phi_*(p) > 0$ and infeasibility is detected.

Table 5.1

$c_1(x) = \sin(x_1^2 + x_2^2) + x_1$

$c_2(x) = \frac{4}{\pi^2}(x_1 + \frac{3\pi}{2})^2 + x_2^2 - 1$

initial guess = $(0, (3\pi/2)^{1/2})$

p	iterations inside linearly constrained optimization routine	$x_*(p)$	$\phi_*(p)$	constraint values at $x_*(p)$
0	14	(-6.3, 0.33)	-5.2	-5.3, 0.1
0.37	2	(-6.3, 0.01)	-2.3	-5.8, 0.1
3.7	1	(-6.0, 0.01)	-0.47	-6.8, -0.36

(feasible point found)

Table 5.2

$c_1(x) = (x_1/2 - 3)\, x_1 + 2x_2 - 1$

$c_2(x) = x_1 + (x_2/2 - 3)\, x_2 + 2x_3 - 1$

$c_3(x) = x_2 + (x_3/2 - 3)\, x_3 + 2x_4 - 1$

$c_4(x) = x_3 + (x_4/2 - 3)\, x_4 + 2x_5 - 1$

$c_5(x) = x_4 + (x_5/2 - 3)\, x_5 - 1$

initial guess = (1, 1, 1, 1, 1)

p	iterations inside linearly constrained optimization routine	$x_*(p)$	$\phi_*(p)$	constraint values at $x_*(p)$
0	1	(2, 0, 0, 0, 1)	-7.5	-5,1-1,1,-3.5
0.37	2	(1.3,0.7,0.3,0.2,0.3)	-4.9	-2.5,-1,-0.7, -0.6,-1.7

(feasible point found)

Table 5.3

$c_1(x) = x_1^2 + 2x_2^2 - 4$

$c_2(x) = x_1^2 + 2x_2 + x_3^3 - 8$

$c_3(x) = (x_1-1)^2 + (2x_2-2^{1/2})^2 + (x_3-5)^2 - 4$

initial guess = (1.0,0.7,5.0)

p	iterations inside linearly constrained optimization routine	$x_*(p)$	$\phi_*(p)$	constraint values at $x_*(p)$
0	4	(0.3,0.3,1.5)	1.7	-3.7, -3.7, 9.1

(concluded infeasible)

Table 5.4

$c_1(x)$ = quadratic with minimum value of 1 at $x = \underline{0} \in R^{10}$

$c_2(x), \ldots, c_{10}(x)$ = quadratic with minimum value of -1 at $x = \underline{0} \in R^{10}$

initial guess = (1,1,...1)

p	iterations inside linearly constrained optimization routine	$x_*(p)$	$\phi_*(p)$	constraint values at $x_*(p)$
0	1	$\sim$(0,0,...0)	-9.0	(1,-1,-1,...,-1)
0.4	1	$\sim$(0,0,...0)	-3.7	(1,-1,-1,...,-1)
4.0	1	$\sim$(0,0,...0)	11.2	(1,-1,-1,...,-1)

6. Summary and directions for further work

We have presented a new class of algorithms (Algorithm 2.3) for finding a feasible point or determining infeasibility of a set of nonlinear inequality constraints

$$c_i(x) \le 0, \quad i = 1, \ldots, m$$

subject to satisfying a set of linear inequality constraints

$$\ell_j(x) \le 0, \quad j = 1, \ldots, k$$

which are known to be feasible (problem (1.2II)). Our class of algorithms consists of solving a series of problems of the form

$$\min_{x \in R^n} \ \phi(x,p) = \frac{1}{p} \sum_{i=1}^{m} w(p \cdot c_i(x)) \qquad (6.1)$$

$$\text{subject to } \ell_j(x) \le 0, \ j = 1, \ldots, k$$

for increasing values of the nonnegative parameter p, where w is a weighting function obeying Definition 2.1. We have shown that in almost all cases (the lone exception being a problem which is exactly feasible, i.e., feasible but with no point where all the constraints are negative) that once p reaches or exceeds a finite value, the answer to (6.1) will either be a feasible point or establish infeasibility of our problem. This may depend, however, on the ability of our algorithm for (6.1) to find the global solution to this problem. An initial computer implementation of our algorithm using $w(y) = e^y - 1$ has successfully solved a wide range of feasible and infeasible problems.

We are actively continuing work on many aspects of this project. Foremost is the improvement of the computer implementation of Algorithm 2.3. The main part of this job is the continued development of a linearly constrained optimization routine which works on nonconvex as well as convex problems and attempts to find the global minimum. We

also intend to experiment with other choices of the weighting function w(y) as indicated in Section 4, and with other strategies for increasing the parameter p including the final one indicated in Section 5. Ultimately we hope to develop reliable and extensively tested software for solving problem (1.2II).

A second remaining aspect of our work is to compare our algorithm with other approaches that can be used to solve the feasibility problem. In particular, we have in mind approaches involving non-differentiable optimization (e.g., the minimax formulation of equation (2.3)) or optimization subject to nonlinear constraints (e.g., the formulation of Bertsekas[1]). We would naturally be interested in comparison with any other algorithm which solves the feasibility problem.

Finally, we would like to extend the work of this paper to cover a broader class of feasibility problems. In particular, we have excluded equality constraints from our consideration so far. While the addition of <u>linear</u> equality constraints causes no change to our theory and very little to our software, how to handle <u>nonlinear</u> equality constraints is far less obvious. Certainly a nonlinear equality constraint $c_1(x) = 0$ can be equivalently expressed as the two inequality constraints

$$\begin{aligned} c_1(x) &\leq 0 \\ -c_1(x) &\leq 0 \ . \end{aligned} \tag{6.2}$$

However, a system of inequality constraints including (6.2) is exactly the type which may give our algorithm trouble, because any feasible point will have to be exactly feasible. The situation may not be as bad as this; in particular, a generalization of the second

part of Example 3.7 shows that if a system of inequality constraints contains only pairs of the type (6.2) and is feasible, then the $x_*(p)$ produced by Algorithm 2.3 will be feasible for any $p > 0$. However, if standard inequality constraints are included as well, then $x_*(p)$ may be feasible only in the limit as $p \to \infty$. Of course, if we stop our algorithm when $\phi_*(p) > 0$ or $x_*(p)$ is within δ of feasibility for some $\delta > 0$ (stopping criteria (3.16)) then it will terminate for finite p on these problems as well, and the practical question is whether p has to grow too large. It will be interesting to see how our computer algorithm performs on systems with equality constraints expressed in form (6.2), and also to see whether any other approaches lead to a better treatment of equality constraints.

Acknowledgments

The author thanks Professors L. Fosdick and L. Osterweil of the University of Colorado at Boulder, who first interested us in this problem in the context of their work in static software validation, and whose NSF grant MCS77-02194 and ARO grant DAAG29-78-G-0046 supported much of this research; R. Lear, who served as a research assistant under these grants and programmed the initial version of our algorithm, as well as providing many helpful suggestions, Professor R. T. Rockafeller of the University of Washington for several helpful discussions at the early stages of this project, and especially for informing us of the basic convexity result in Lemma 2.2 which is of fundamental importance to this work; and Professor M. J. D. Powell of Cambridge University for his remarks about using our algorithm on problems with equality constraints.

References

[1] D. P. Bertsekas,"Nondifferentiable optimization via approximation," Mathematical Programming Study 3 (1975) 1-25.

[2] C. Charalambous and A. R. Conn, "An efficient method to solve the minimax problem directly," SIAM Journal on Numerical Analysis 15 (1978) 162-187.

[3] L. Clarke, "A system to generate test data and symbolically execute programs," IEEE Transactions of Software Engineering 2 (1976) 215-222.

[4] G. B. Dantzig, Linear Programming and Applications (Princeton University Press, Princeton, 1963).

[5] A. V. Fiacco and G. P. McCormick, Nonlinear Programming: Sequential Unconstrained Minimization Techniques (John Wiley, New York, 1968).

[6] R. Fletcher, "An algorithm for solving linearly constrained optimization problems," Mathematical Programming 2 (1972) 133-165.

[7] P. E. Gill and W. Murray, eds., Numerical Methods for Constrained Optimization (Academic Press, London, 1974).

[8] D. Goldfarb, "Extensions of Davidon's variable metric algorithm to maximization under linear inequality and equality constraints, SIAM Journal on Applied Mathematics 17 (1969) 739-764.

[9] D. G. Luenberger, Introduction to Linear and Nonlinear Programming (Addison-Wesley, Reading, Mass., 1973).

[10] D. A. Pierre and H. J. Lowe, Mathematical Programming via Augmented Lagrangians (Addison-Wesley, Reading, Mass., 1975).

[11] S. M. Robinson, "Extension of Newton's Method to nonlinear functions with values in a cone," Numerische Mathematik 19 (1972) 341-347.

[12] R. T. Rockafellar, Convex Analysis (Princeton University Press, Princeton, 1970).

[13] I. Zang, private communication.

[14] I. Zang, "A smoothing-out technique for min-max optimization," Technical Report, Tel-Aviv University (Tel-Aviv, 1979).